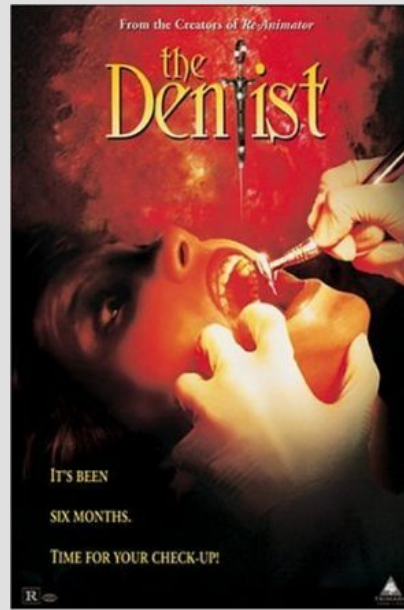




Gold or Amalgam

Sharpen your sixth sense detecting weak fixes,
and stronger code surgery

1

- Working daytime for Red Hat, busy with Java bugs
- Did talks on Security and RE topics on several conferences before (CSW, PacSec, HITB, SyScan, Blackhat, ...)
- Most vulnerabilities published until end of 2009 were java-centric or web application flaws
- The results presented in this talk presents research done „after hours“, hence this is no official Red Hat talk

The Speaker

2

- The talk describes observations I made while finding, reporting, and verifying security bugs
- In 2009 I thought like there must be some shorter way to find software bugs than by reading code
- Ideally not focussed only on java (doesn't win at CFPs) but still cross-OS attack surface

Motivation

3

- Software Fixes (Towards Defining a Weakness property)
- Detection Strategy (Fuzzing approaches)
- Knowledge is your arsenal (Specifications, Protocols)
- Tool-based support
- Examples (Chrome, Windows, Firefox, JDK, ...)

Today's walkpath

4

- Surface fixes
 - Fast delivery ,
 - Easy workflow: first draft = shipped version
 - helps to the users quiet
- Root causes fixes
 - Take their time
 - Discussion process: multiple patch iterations
 - Risks for users getting owned
 - Risk for „riots“ in user base not getting early patch

Working term: Software fix 5

- are typically weak fixes
 - do not fix root cause
 - may be as isolated as just blocking single instance of a bug pattern
 - Can be caused by:
 - incomplete triage , missing knowledge of broken component,
 - hidden functionality in binary not obvious in source (macros, templates, generators), optimization magic
 - No source, even worse obfuscated binaries
 - In the long cause:
 - Code decay

Surfaces Fixes

6

- Allow to renovate decayed code areas
 - fix root cause,
 - refactor larger areas of code
 - Introduce interception patterns, validation hooks
 - May be necessary because:
 - Calls into underlying libs have proven to be a can of worms
 - Fixing individual bugs is in the long run more time-intensive than re-doing it right
 - Coder with intense knowledge of component found out new threat pattern or is concerned about defense-in-depth strength,

Structural Fixes

7

- Programmers view
 - A chess board has 8 x 8 fields, a chess set contains King, Queens, Bishops, etc.
- Source code
 - `int[] Board = new int[64];`
 - `a[3]=King_Black;`
- Compiled code
 - `Mov /* mal gcc anwerfen */`
- Machine's view
 - No clue of chess
 - Allows ghost fields like `a[103]` , throughout all accessibly memory

Semantical Layers

8

- Keep in mind every measurement involves subjectivity
[<http://ptjournal.apta.org/content/69/7/577.full.pdf>]
- But every theory needs a base assumption in order to start
- Bug Proximity determined as source code proximity
- Bug Proximity as crash dump proximity (registers, stack dump) , all we have for closed source

Proximity between bug candidates

9

- We define a strong fix is one that has minimal proximity to an open bug (normalization =1.0)
- The larger the proximity the weaker the fix (normalization=0.0)
- Of course, the world isn't black and white, so bugs are typically somewhere in between this continuum

Proximity between bug candidates

10

- A crash in the Apple Type Services (ATS)

National Vulnerability Database
Automating vulnerability management, security measurement, and compliance checking

Sponsored by
DHS National Cyber Security Division/US-CERT

NIST
National Institute of Standards and Technology

National Cyber-Alert System

Vulnerability Summary for CVE-2011-0177

Original release date: 03/23/2011

Last revised: 03/24/2011

Source: US-CERT/NIST

Overview

Multiple buffer overflows in Apple Type Services (ATS) in Apple Mac OS X before 10.6.7 allow remote attackers to execute arbitrary code via a document that contains a crafted SFNT table in an embedded font.

Impact

CVSS Severity (version 2.0):
CVSS v2 Base Score: 6.8 (MEDIUM) (AV:N/AC:M/Au:N/C:P/I:P/A:P) (legend)

IVD contains:

- 6092 [CVE Vulnerabilities](#)
- 169 [Checklists](#)
- 212 [US-CERT Alerts](#)

Example: CVE-2011-0177

11

- The original ATS reproducer was found by iterating with the font fuzzer over the qlmanage tool
- The first bug was reported
- To evaluate if it was a shallow or deep fix, basically the same test was run with different font files and longer fuzzing cycles

```
exception=EXC_BAD_ACCESS:signal=11:is_exploitable=yes:instruction_disassembly=movdqa
%xmm0,CONSTANT(%rdi,%rcx):instruction_address=0x00007fffffe0092e:
access_type=write:access_address=0x0000000106affff0:
```

A good fix makes finding similar bugs harder

12

	A	B	C	D	E	F	G	H
1	case	type	sig	exp	Opcode	crash address	access	Address
2	ado449	EXC_CRASH	6	yes		0x00007fff8		0x0000000000000000
3	ado817	EXC_CRASH	6	yes		0x00007fff8		0x0000000000000000
4	ado988	EXC_CRASH	6	yes		0x00007fff8		0x0000000000000000
5	adobe_re	EXC_BAD_ACCESS	11	yes	movdqa %xmm3,CONSTANT('0x00000000		write	0x00000000fcfd2000
6	adobe_re	EXC_BAD_ACCESS	11	yes	movdqa %xmm2,CONSTANT('0x00000000		write	0x00000000ff327000
7	bmo1051	EXC_BAD_ACCESS	11	yes	movdqa %xmm2,CONSTANT('0x00007fffff		write	0x00000001069ffff0
8	bmo1056	EXC_BAD_ACCESS	10	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001000fdff0
9	bmo1067	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
10	bmo1098	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001002ffff0
11	bmo1099	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
12	bmo1137	EXC_BAD_ACCESS	10	yes	movdqa %xmm2,CONSTANT('0x00007fffff		write	0x0000000103ffffff0
13	bmo1167	EXC_BAD_ACCESS	10	yes	movdqa %xmm2,CONSTANT('0x00007fffff		write	0x00000001000fdff0
14	bmo1179	EXC_BAD_ACCESS	10	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001000fdff0
15	bmo1201	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001002ffff0
16	bmo1204	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001073ffff0
17	bmo1210	EXC_BAD_ACCESS	11	yes	movdqa %xmm0,CONSTANT('0x00007fffff		write	0x0000000105effff0
18	bmo1214	EXC_BAD_ACCESS	11	yes	movdqa %xmm0,CONSTANT('0x00007fffff		write	0x00000001002ffff0
19	bmo1259	EXC_BAD_ACCESS	10	yes	movdqa %xmm0,CONSTANT('0x00007fffff		write	0x00000001000fdff0
20	bmo1269	EXC_CRASH	6	yes		0x00007fff8		0x0000000000000000
21	bmo1275	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
22	bmo1279	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
23	bmo129	EXC_BAD_ACCESS	11	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001058ffff0
24	bmo1316	EXC_BAD_ACCESS	10	yes	movdqa %xmm0,CONSTANT('0x00007fffff		write	0x00000001000fdff0
25	bmo1321	EXC_BAD_ACCESS	11	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001002ffff0
26	bmo1364	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
27	bmo1369	EXC_BAD_ACCESS	10	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001000fdff0
28	bmo1376	EXC_BAD_ACCESS	11	yes	movdqa %xmm0,CONSTANT('0x00007fffff		write	0x00000001057ffff0
29	bmo1453	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001002ffff0
30	bmo1458	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001002ffff0
31	bmo1570	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x0000000103ffffff0
32	bmo1585	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001065ffff0
33	bmo1592	EXC_BAD_ACCESS	10	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001000fdff0
34	bmo1593	EXC_BAD_ACCESS	11	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001002ffff0
35	bmo1594	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001001002ffff0
36	bmo1595	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001001000fdff0
37	bmo1596	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001001068ffff0
38	bmo1618	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
39	bmo1643	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001000fdff0
40	bmo1645	EXC_BAD_ACCESS	11	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x0000000105dffff0
41	bmo1646	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001002ffff0
42	bmo1647	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x0000000106ffffff0
43	bmo1648	EXC_BAD_ACCESS	10	yes	movdqa %xmm1,CONSTANT('0x00007fffff		write	0x00000001002ffff0
44	bmo1800	EXC_BAD_ACCESS	11	yes	movdqa %xmm3,CONSTANT('0x00007fffff		write	0x00000001002ffff0

A good fix makes finding similar bugs harder

KNOWLEDGE BOOSTS YOUR ARSENAL (SPECIFICATIONS, PROTOCOLS)

14

- Successful detection of weak fixes via fuzzing requires
- Know the specs
- Write parsers
- Learn to identify invariants to get into vulnerable areas
- Example : Android dexdump CVE-2011-1001

Get to know the details!

15

- APSB 11-02 of February 2011 fixed CVE-2010-0577
 - Fonts in Flash since the early days, tags DefineFont , DefineFont2/3 used flash-specific glyph shape tables
 - Flash 10 introduced the DefineFont4 (id=91) tag , allowing to embed complete Compact font file (CFF) structures
- Used **zzuf** and my flash parser to go for range fuzzing

```
python parseflash.py clays/clay.swf | grep DefineFont
115549:115555:137887:91:DefineFont4(10):22332:{'fontname':
'Windsong', 'reserved': 0, 'fontFlagsBold': 0, 'fontFlagsItalic':
0, 'fontdata': 'OTTO\x00\n\x00\x80\x00\x03\x00 CFF \x95\xa3B
\xb8\x00\x00\x00\xac\x00\x00N\x00S/2 .....
```

Parse & Fuzz, Flash R Us

16

- Dexdump is a tool to dump code and structure of android executables (dex)
- Dump fuzzing doesn't get you far , as it contains checksums
- To reach beyond the surface code, you need to mimic the original checksumming code as finishing touch with every fuzzing test case

Correct the Chksums

Android SDK CVE-2011-1001 17

```
import sys
import zlib
import struct
z = sys.argv[1]
k = file(z,"rb").read()
adler = zlib.adler32(k[12:]) & 0xffffffff
y = struct.pack("I",adler)
l = k[0:8]
m = k[12:]
l = "%s%s%s" % (l, y,m)
f = open("%s.patch.dex" % z,"wb")
f.write(l)
f.close()
```

Correct the Chksums

Android SDK CVE-2011-1001 18

DETECTION STRATEGY (FUZZING APPROACHES)

19

- Zzuf
 - Author: Sam Hocevar, License: DWTFYWTPL
 - Xor Bit flip fuzzing
 - transparent proxy
 - Highly configurable fuzz parameters (seed, density, range, ...)
 - Seed not reproducible when java is called within zzuf
 - Use shell script wrapper to separate processes

Fuzzer

20

- Honggfuzz
 - Author: Robert Swiecki, License: Apache 2
 - byte level fuzzing is default
 - Integrated crash analysis (ptrace)
 - Only a few fuzzing parameters
 - Version 0.1 (handling of longer runs difficult)
 - Interesting pre-bucketing via crash sig in filenames

Fuzzer

21

- Parts of JDK implemented in native C code
 - Image parsers
 - Sound parsers
 - Font rendering
 - Color Profiles
 - Class files
- You leave the safe world of guarded array access , managed pointers and fool-proof memory mgmt
 - Buffer overflows
 - Heap corruption

Why fuzz Java

22

- Neither zzuf nor Honggfuzz are optimal for fuzzing java
- Honggfuzz stops longer runs on OOM condition (not usable for batches)
- General
 - Run JVM with `-Djava.awt.headless=true` to avoid Swing effects (app not shutting down, slow startup times)
 - In rare cases the reproducers are unusable, as they only work when run under the fuzzer

Fuzzer

23

- Why Java Fuzzing
 - Finding exploitable bugs in Java System Libraries is time-consuming
 - requires deep understanding about security architecture,
 - Reduced amount of hookable code paths for trusted-method chaining, privileged deserialization
- Why not choose the easy way: Fuzzing JNI-Code
 - JNI, Javas calling convention into C / C++
 - No security manager, welcome back memory corruption and overflows
- Basic Approach:
 - Bypass integrity constraints in order to crash in native routines
- Generate Fuzzed Mutants with in the JVM
 - Tools to fuzz the JVM
 - In-JVM fuzzing

Java-In-Process Fuzzing

24

- IN-JVM fuzzing
 - Works on byte[] – Buffers
 - Emulates zzuf range and ratio parameters
 - Only restart of JVM once crash happened (save time)
 - To isolate test case create standalone reproducer
 - Works in non-headless mode too

Fuzzer

25

```
static int[] pot= new int[]{1,2,4,8,16,32,64,128};

public static RetVal fuzz(byte[] arr,int seed, double ratio) {
    Random rand = new Random(seed);
    RetVal r = new RetVal();
    int size = arr.length;
    int size8= size * 8;
    int anz = (int) (size * ratio) ;
    r.changes = anz;
    r.b = arr.clone();
    if (ratio!= 0.0) {
        for (int i = 0 ; i < anz; i++) {
            int pos = rand.nextInt(size8-1);
            int posby = pos / 8 ;
            int posbt = pos % 8;
            byte od = r.b[posby];
            byte newod = (byte) (od ^ pot[posbt]);
            r.b[posby]=newod;
        }
    }
    return r;
}
```

Selected Example: A Java Fuzzer

26

- Currently Generation 3
 - optional Lightweight web server (python CGIHTTPServer)
 - Fuzzing engine is ported to javascript
 - DOM and CSS is updated on the fly
 - Fuzzed content replaced in <div> element, no page reload
 - Ability to dump simplified reproducer case
 - Ability to use different html templates (to test interaction with CSS effects, shadows, text stroke, etc.)
 - Will be released under GPL soon

A Font Fuzzer

27

TOOL-BASED ANALYSIS

28

- Typical categories of Bugs with Dynamically allocated memory
 - Invalid allocations (app not checking for null)
 - Invalid frees (app freeing the same block multiple times)
 - Out-Of-Bound-reads
 - Out-Of-Bound-writes
- Electric Fence, GuardMalloc, MALLOC_CHECK_=3

Sensors for failed Dynamic Memory allocation

29

- CrashWrangler
 - On OSX , gives neat „exploitable: yes/no“ notice by checking regs and crash instruction
 - eases discussion about security relevance of report
- GDB
 - The choice on Linux,
 - General to dig deeper on *X operating system
- WinDbg
 - Relatively transparent debugger on W32
 - !MSEC.exploitable analyses regs and crash instruction

Crash triage

30

- Identify common pattern in the crash dumps
 - Registers, Stackdumps
 - Similarity metrics
- Apply spreadsheet magic
 - Data collection
 - Mine your crashes, cluster results
 - Extract common patterns
 - Finetune coverage of tests (extended API usage)
- Reproducer conservation
 - Raw File
 - Embed in Web page (Font, Image)

Crash bucketing

31

The larger picture

32

EXAMPLES (CHROME, WINDOWS, FIREFOX, JDK, ...)

33

- SEGV processing Midi Soundbanks
- A vulnerability was discovered in the handling of MidiSystem.getSoundbank (CVE-2010-3572)
- Method used:

```
for i in {1..100} ; do echo $i ; zzuf -s$i
-r 0.004 <soundbank-min.gm >/tmp/sb$i ;
CW_CURRENT_CASE=sb$i exc_handler java -
Djava.awt.headless=true DisplaySoundbank /
tmp/sb$i ; done
```
- On crash collect data, later mine for suspicious crashes

Selected Example: CVE-2010-3572

34

- SEGV processing Midi Soundbanks
- A vulnerability was discovered in the handling of Midi soundbanks (CVE-2010-4473)

- Method used:

```
for i in {1..100} ; do echo $i ; zzuf -s$i
-r 0.004 <soundbank-min.gm >/tmp/sb$i ;
CW_CURRENT_CASE=sb$i exc_handler java -
Djava.awt.headless=true
DisplaySoundbankExt /tmp/sb$i ; done
```

- Same Setup, slightly extended API use

Selected Example: CVE-2010-4473

35

- July 2010, Chrome Bug #48283 (CVE-2010-2897)
 - Found Dumb fuzzing Chrome 5 on Windows XP/SP3, making sure KB979559 font fix applied
 - After a longer fuzzing run , the machine suddenly **rebooted**,
 - a retry still did, so the bug looked stable
- Google security team investigated this to be a bug in windows ATMFd (Adobe Type Manager) stumbling over broken CFF table offset sizes
- To protect Chrome users from this windows bug , OTS in Chrome 6 was hardened to catch broken offsets
- However it was not fully fixed yet

Chrome bugs & reboots

36

- August 2010, Chrome Bug #51070 (CVE-2010-3111)
 - After cr#48283 was allegedly fixed , so I tried to verify
 - Slightly modified test case, with other fonts too on XP/SP3
 - And again, the **reboots**, a either incomplete fix or new bug
- Same game: Google security team investigated this to be a different bug
 - in windows ATMFD, having problems with malicious font hinting code using an oversized stack
- To protect Chrome users from this windows bug OTS was hardened to block those malicious hinting information to harm
- Microsoft confirmed both bugs to fix at a later point in time (Dec 2010)

Chrome bugs & reboots

37

September 2010, Mozilla Bug #583520 (CVE-2010-2770)

- Fonzfuzzer was able to find more real-life bugs on SOX
- This time primarily on OSX
- After a while crashwrangler reported a double-free issue
- Firefox security team confirmed this to be an exploitable bug and refined the patch over multiple iterations

```
Faulty glyph (id:38) outline detected - replacing with a space/null
glyph - in memory font kind
Fri Jul 30 12:20:24 maeckes2.local firefox-bin[10483] <Error>:
CGBitmapContextInfoCreate: unable to allocate 10584 bytes for bitmap data
objc[10483]: FREED(id): message autorelease sent to freed object=0x1f2de
[...]

---
exception=EXC_BAD_INSTRUCTION:signal=4:is_exploitable=yes:
instruction_disassembly=:instruction_address=0x0000000097db24b4:
access_type=:access_address=0x0000000000000000:
Illegal instruction at 0x0000000097db24b4, probably a exploitable issue.
```

Fi

December 2010, Mozilla Bug #583520 (CVE-2010-3768)

- I gave **Generation 2** to Mozilla security team, they found out a series of numerous other bugs
- Additionally I reported the following ‘invalid write’ one:

```
exception=EXC_BAD_ACCESS:signal=11:is_exploitable=yes:instruction_disassemb  
ly=movl %eax, (%esi):instruction_address=0x000000009011404d: access_type=  
write:access_address=0x00000000fedd02b4:  
Crash accessing invalid address.
```

- And could not resist the following question:
 - *“Is your future strategy to handle the font bugs case wise, or probably introducing stricter acceptance rules via a (better be sandboxed) font sanitizer ?”*
- Mozilla added protection by integrating OTS(Open Type Sanitizer)

Firefox bugs: Strong fix

39

CONCLUSION

40

- Fuzzing is a powerful technique to detect bugs
- We showed how to embed the raw fuzzing procedure in a workflow to tweak result output
- Every scenario has a different mix between stock fuzzers and self-written finishing tools

Resume

41

- Thanks for working with me, fixing the reported bugs:
 - Adobe PSIRT
 - Apple Security Team
 - Android Security Team
 - Chrome Security Team
 - Microsoft Security team
 - Mozilla Security Team
 - My colleagues at Red Hat Security Response Team
 - Oracle Java Security Team

Acknowledgements

42

Questions ?

Marc /at/ illegalaccess /dot/ org

43